Restricted Slow-Start for TCP

William Allcock[1,2], Sanjay Hegde[3], Rajkumar Kettimuthu[1,2]

[1]Argonne National Laboratory, Argonne, IL 60439, USA
[2]The University of Chicago, Chicago, IL 60637, USA
[3]California Institute of Technology, Pasadena, CA 91125, USA
{allcock, kettimut}@mcs.anl.gov, hegdesan@caltech.edu

Abstract

In network protocol research a common goal is optimal bandwidth utilization, while still being network friendly. The drawback of TCP in networks with large bandwidth-delay products due to its AIMD based congestion control mechanism is well known. The congestion control algorithm of TCP has two phases namely slow-start phase and congestion-avoidance phase. Many researchers have focused on modifying the congestion avoidance phase of the algorithm. In this work, we propose a modification to the slow-start phase of the algorithm to achieve better performance. Restricted slow-start algorithm is a simple sender side alteration to the TCP congestion window update algorithm.

1. Introduction

TCP was originally defined in RFC 793 [1], and several enhancements have been proposed to TCP since then. The congestion control algorithm [2] of TCP has two phases namely slow-start phase and congestion-avoidance phase. With slow start, the sender window begins at one segment and is incremented by one segment every time an acknowledgment is received. This opens the window exponentially: send one segment, then two, then four, and so on. With congestion avoidance, the sender window is incremented at most one segment each round-trip time, regardless of how many acknowledgments are received in that round-trip time. The congestion control algorithm starts with the slow-start phase. Whenever congestion is detected, it reduces the sender window to half of its value and enters congestion avoidance. The current slow-start procedure can result in increasing the sender window by thousands of segments in a single round-trip time for networks with large bandwidth-delay products. Such an increase can easily result in thousands of packets being dropped in one round-trip time. This is often counter-productive for the TCP flow itself, and is also hard on the rest of the traffic sharing the congested link. In this work, we propose a modification to the slow-start procedure to solve this problem and improve the network utilization.

2. Background and motivation

Congestion occurs when the traffic offered to a communication network exceeds its available transmission capacity. But congestion events are not just pertained to congestion in the network. In some operating systems (for example: Linux), congestion events (send-stalls) are generated due to the saturation of several soft components such as buffers and queues in the host. Though these are resource constraints at the sending host and are not in any way indicate of congestion in the network, Linux TCP treats these events in the same way as it would treat the network congestion. The impact of these send-stall events was reflected in the demo that we conducted at IGrid2002. Further analysis revealed that these congestion events (send-stalls) are generated in the slow-start phase rather in the congestion avoidance phase. We propose a control theory approach that appropriately paces the TCP sender during the slow-start phase to avoid the saturation of soft

component such as network interface queue. Even though there have been proposals to increase the size of these soft components to overcome this problem, deployment of these solutions revealed that still a considerable amount of available bandwidth goes unutilized. Also, increasing the size of the soft components increases the memory usage. We aim at improving the end-to-end bandwidth utilization without increasing the memory usage at the host.

3. Proposed Scheme

We use a PID control algorithm [3] to determine the rate of increase during the slow-start phase. In the PID control approach, the gain is calculated using a first order differential equation. The controller gains are configurable. The 90% of the maximum value of the network interface queue (IFQ) size is used as the set point and the current value of the IFQ is used as the process variable in the controller. The controller compares the process variable (current IFQ) to its set point (max IFQ) and calculates the error. Based on the error (E), a few adjustable settings and its internal structure, the controller calculates an output that determines the new value of the sender window. The PID transfer function used is

$$K_p * (E) + 1/T_i \int_0^t (E)\, dt + T_d * d(E)/dt)$$

We use Ziegler Nichols Tuning Method [4] to calculate the PID parameters (K_p, T_i and T_d). A brief description of the method is as follows:

- Select proportional control alone
- Increase the value of the proportional gain until the point of instability is reached (sustained oscillations), the critical value of gain, K_c, is reached.
- Measure the period of oscillation to obtain the critical time constant, T_c.

Once the values for K_c and T_c are obtained, the PID parameters are calculated as follows: $K_p = 0.33\ K_c$, $T_i = 0.5\ T_c$, and $T_d = 0.33\ T_c$.

4. Experimental Results

Our scheme is implemented in a 2.4.19 linux kernel and the performance is evaluated through experiments conducted over a 100 mbps link between Argonne National Laboratory and Lawrence Berkeley National Laboratory, a round-trip time of 60 ms. We use web100 [5] to get detailed statistics of the TCP state information. Preliminary results show that our scheme is able to achieve 40% improvement in throughput compared to the standard TCP. Figure 1 compares the cumulative send-stall signals over time in modified TCP with that of the standard Linux TCP.

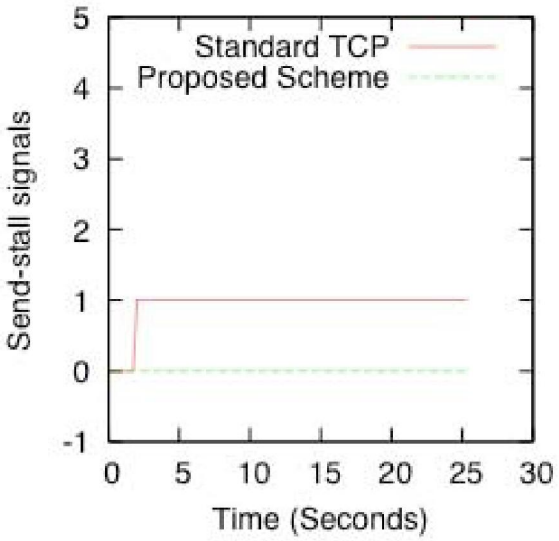

Figure 1: Comparison of send-stall signals in the standard Linux TCP and the modified TCP

Acknowledgments

This work was supported by the University of Chicago under NSF Grant #SCI-0414407, and the U.S. Department of Energy under Contract W-31-109-ENG-38.

References

[1] J. Postel, "Transmission Control Protocol," RFC-793, September 1981.

[2] M. Allman, V. Paxson, W. Stevens, "TCP Congestion Control," RFC-2581, April 1999

[3] J.P. Gerry, "A Comparison of PID Control Algorithms," Control Engineering, pp 102-105, March 1987.

[4] J.G. Ziegler, N.B Nichols, "Optimum settings for automatic controllers," Trans. ASME, pp. 759-768, 1942.

[5] http://www.web100.org/